OPERATING SYSTEMS:

Lesson 10:
Virtual Memory

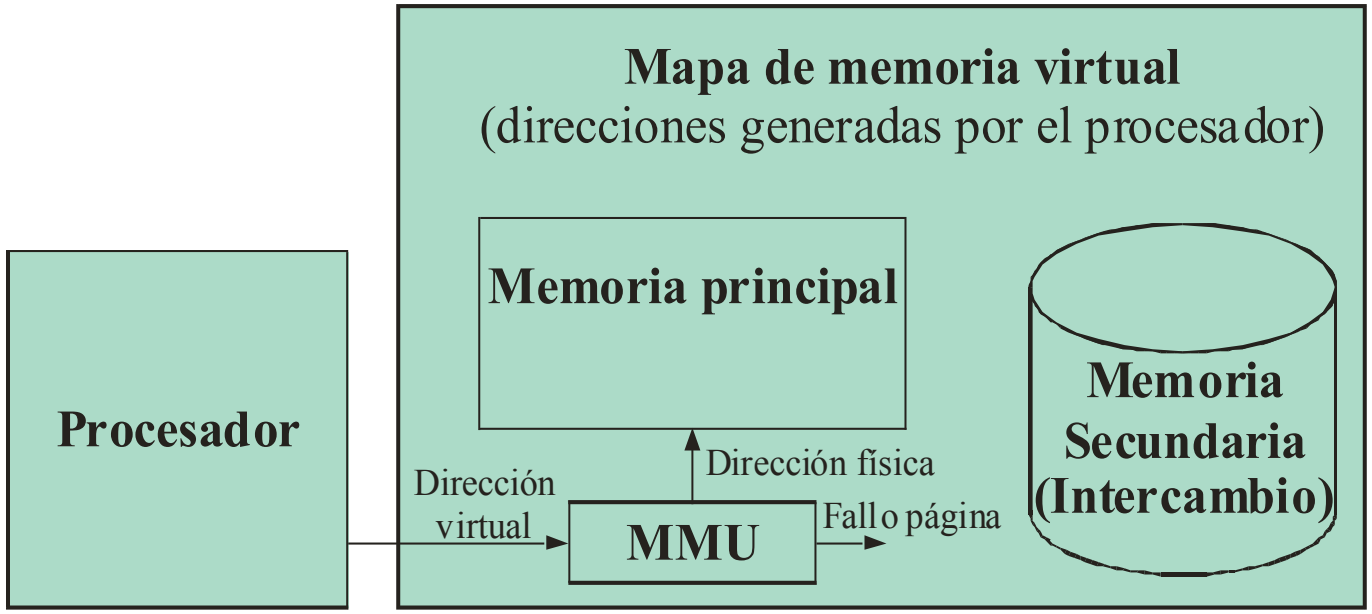
Jesús Carretero Pérez
David Expósito Singh
José Daniel García Sánchez
Francisco Javier García Blas
Florin Isaila

Virtual memory

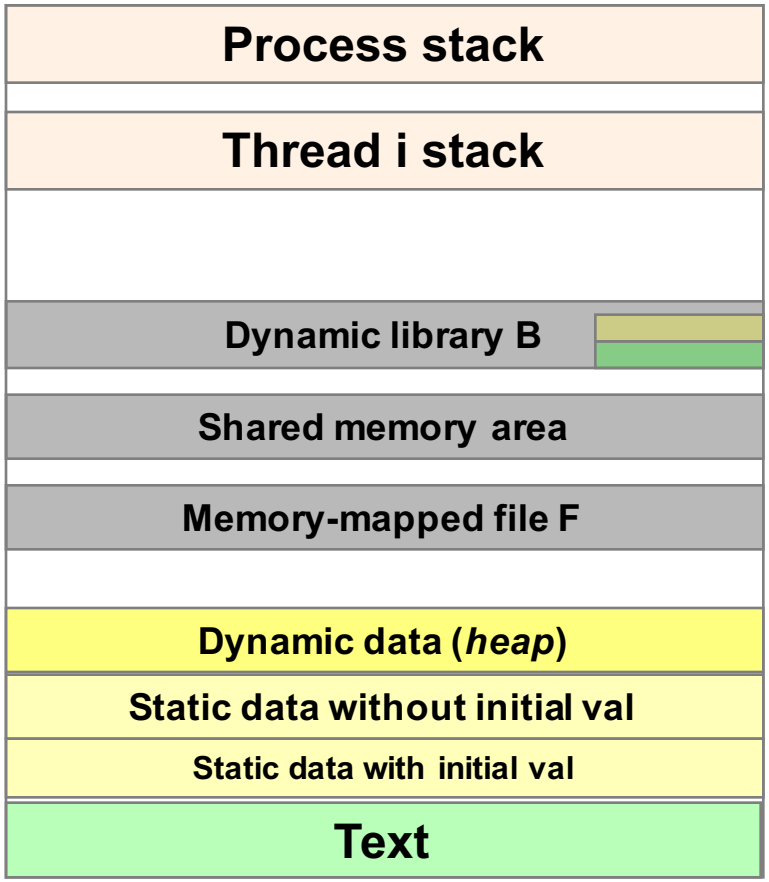
- Virtual memory – separation of user logical memory from physical memory.
  - Only part of the program needs to be in memory for execution
  - Logical address space can therefore be much larger than physical address space
  - Allows address spaces to be shared by several processes (e.g. shared libraries)
  - Allows for more efficient process creation
- Memory Hierarchy : Several storage levels
- Virtual memory secondary
  - Transfers from/to RAM to secondary devices (swap)
- Based on segment and/or pages:

Virtual memory

- Provide automatic management of the part of the extended memory hierarchy formed by the levels of main memory and disk



Example of Virtual Memory Map



- 2^32
- 2^64
- Logical address space larger than physical memory

Virtual memory

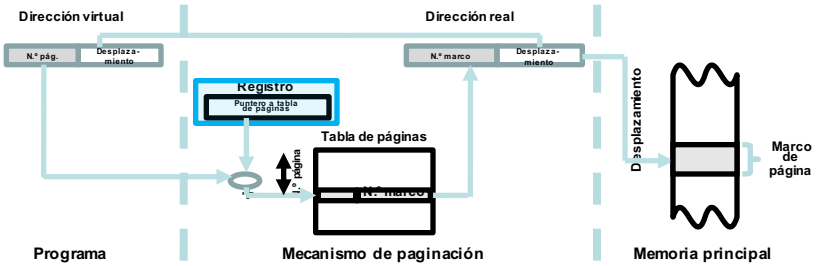
- Benefits:
  - Process protection and reallocation
  - Increased multiprogramming index
    - ¡Possibility of "hyperpagination" [*trashing*]!
  - Execution of programs larger than RAM.
- The process only sees virtual addresses.
- Process virtual map supported on RAM and secondary memory (*swap*).
- No need to have the whole process loaded in memory =>
  - More processes in RAM means more efficient usage of the processor.
  - A process may be bigger than the RAM.
  - Faster booting.
- **Resident set:** part of the process loaded in RAM.

VM requirements

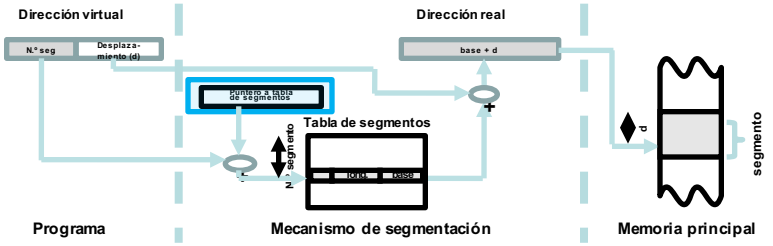
- Needs secondary storage support.
- Virtual memory larger than Physical memory
  - Faults in memory accesses
- Page fault
  - Trap to the OS
- After the trap, the program must continue from the instruction that caused the page fault.
  - MOV (R1), (R2) => three *posible* page faults.

Virtual Memory implementations

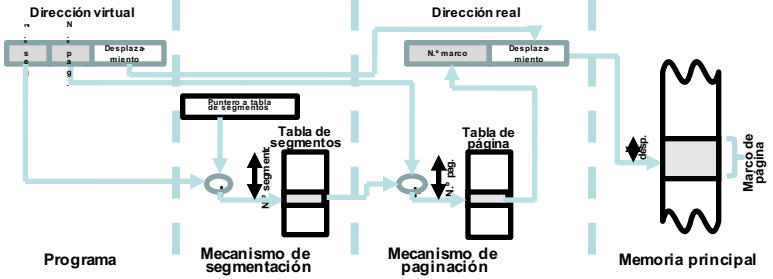
- Paging



- Segmentation



- Paged segmentation



Virtual memory: paging

- Processes exhibit proximity of references
- Virtual memory secondary
  - Transfers from/to RAM to secondary devices (swap)
- Based on pages:
  - Nonresident page is marked absent
  - Dir saved of the swap block that contains it
- Disk → RAM memory: on demand
  - Access page non-resident -> Page fault.
  - OS reads disk page.
- Main memory → disk: expulsion
  - No space to bring page to RAM
  - A resident page is ejected (replaced)
  - OS writes ejected page to disk

Paging

- **Hardware** [¡Mandatory!] more sophisticated
- Allocation unit: pagina: **page**
- Memory map divided in pages
- **Physical memory** divided in page frames
  - Frame size = page size
- **Page table** (PT) per process:
  - Maps each page with the container frame
- **MMU** uses PT to translate logical addresses
- OS provides PT per process and notifies the MMU teh PT to be used for each process

Paging (II)

- PT entry:
  - Protection (RWX)
  - "Resident Bit": Present page (**P**) or absent **(A**)
- Internal fragmentation
  - For each memory zone, only the part of the last frame allocated can be unused. Other are always full.
- Meet requirements?
  - Independient mem spaces: Each process a PT.
  - Protection: Bit in PT.
  - Memory sharing: Pages assciated to the same physical frame
  - Regions: From init one page to limit.
    - Access to page level
    - Space allocated on demand
  - Maximum performance
    - Access to page level easy and fast.
    - Problem: PT can be huge!

Program execution

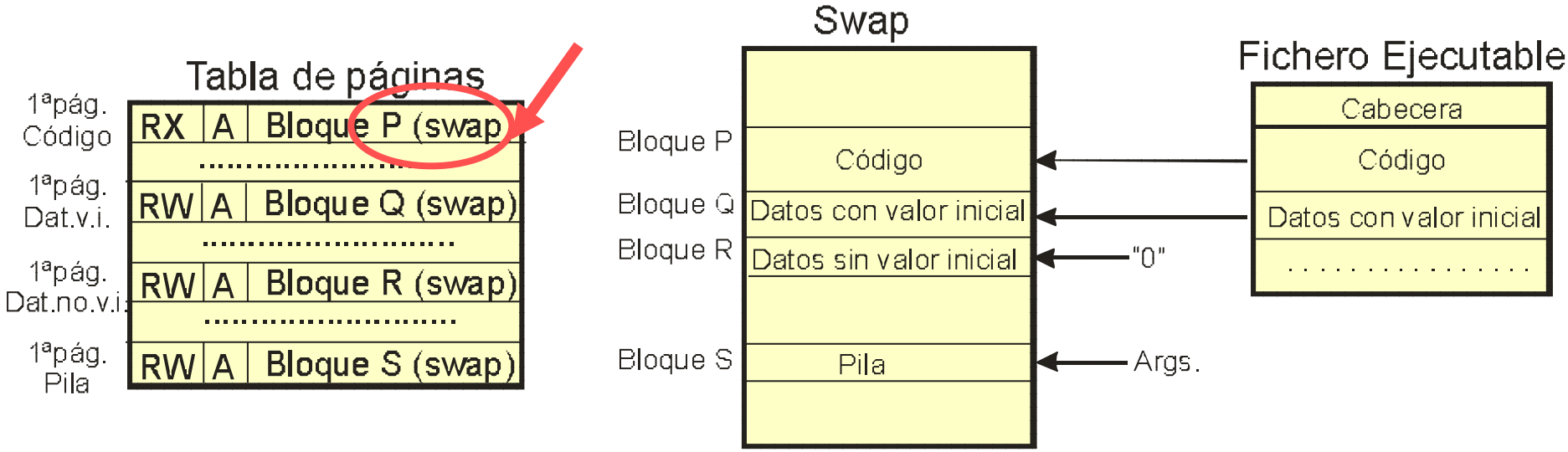
- Creation of regions from the disk exec file
- No spece allocated in RAM memory
- No data loaded in RAM memory
  - Paging on demand
- Each PT entry filled with control bits:
  - Protection: depending on region
  - Absent/present
  - Swap dir of the page
    - This address depends on swap management

Swap space management

- *swap* space allocation when a region is created
- Two alternatives: with and without preallocation
- Region creation **with preallocation**
  - *Swap* space allocated for each region
  - Executbale contents copied to swap pages
  - Pages come to RAM from swap on demand
  - If preempted, space is already allocated
- Region creation **without preallocation**
  - *Swap* space not allocated for each region
  - Pages come to RAM from disk exec file on demand
  - If preempted, space is allocated in *swap* (if page "dirty")
  - *This option is more popular currently*
- Shared regions usually are like any other

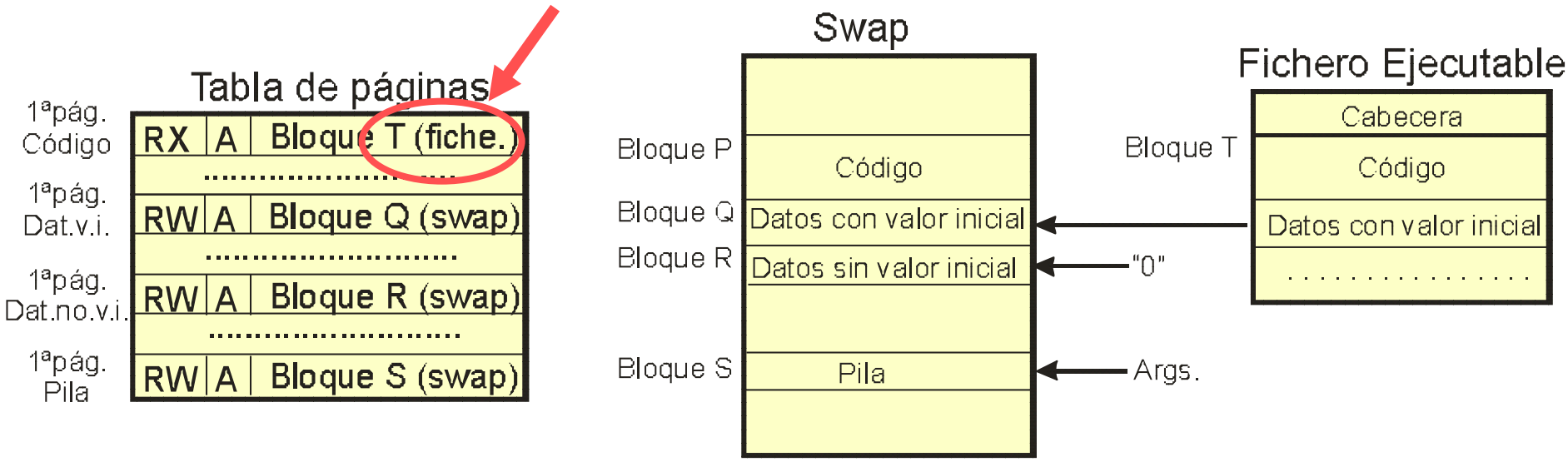
Swap preallocation

- OS allocates in advance swap space for regions
- Copy from exec file to swap:
  - Code: from exec file
  - Data with initial value: from exec file
  - Stack: program arguments
  - Data without initial value: fill to 0
- PT entries are references to blocks in *swap*



Allocation on-demand

- Pages come to RAM from disk exec file on demand
  - Code, data with initial values, …
- Code regions: shared with exec file (only read)
- Swap not used for static/read only data
- Dinamic data and stack allocated in swap on-demand
  - Initially: small regions

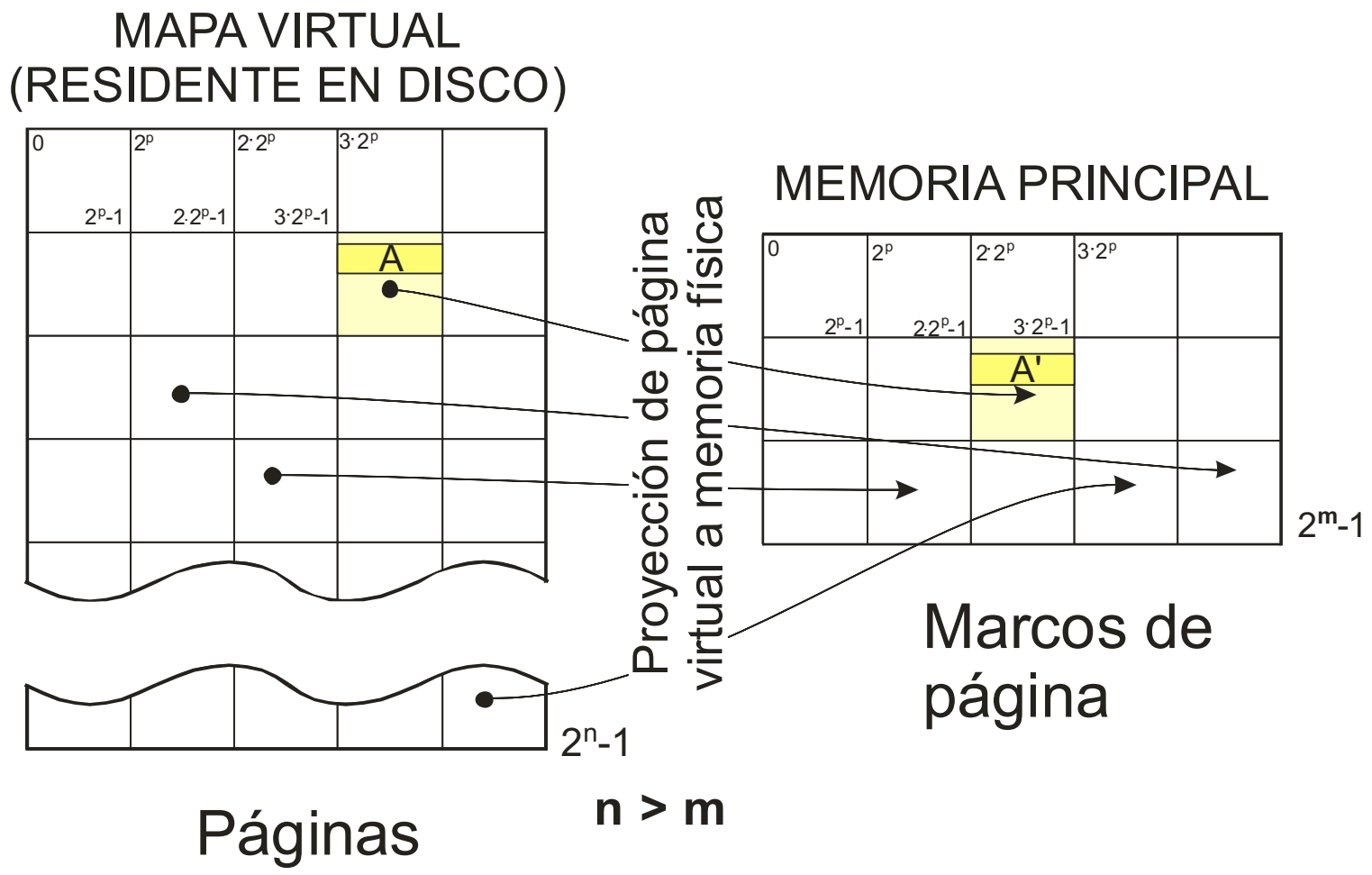


Allocation on-demand

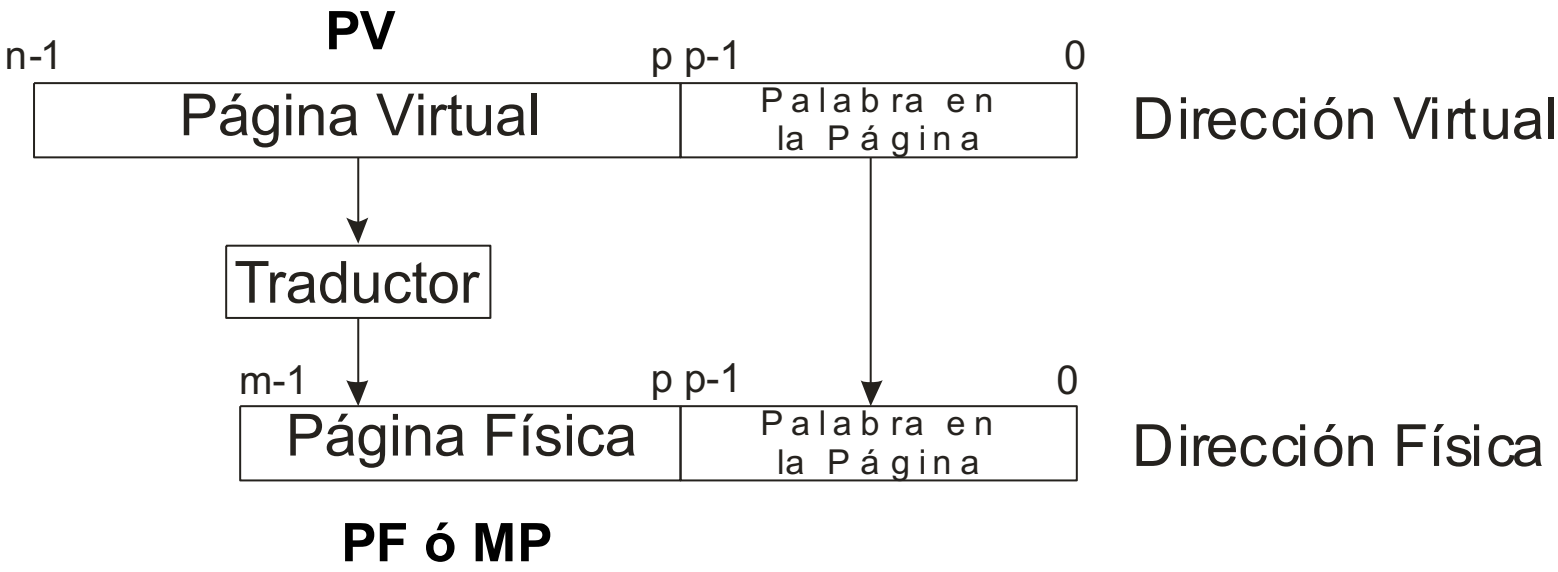
- PT entries: references to blocks of secondary storage: exec file and swap:
  - Code: blocks of exec file
  - Data with initial value: blocks of exec file
  - Stack: swap block with program arguments
  - Data without initial value: fill to 0 in swap blocks

- First time a “dirty” page is preempted, swap space is allocated and page is copied to swap.
  - Except if it is part of a shared region using a file -> updated in the file.

Paging: virtual to physical



Paging: address translation



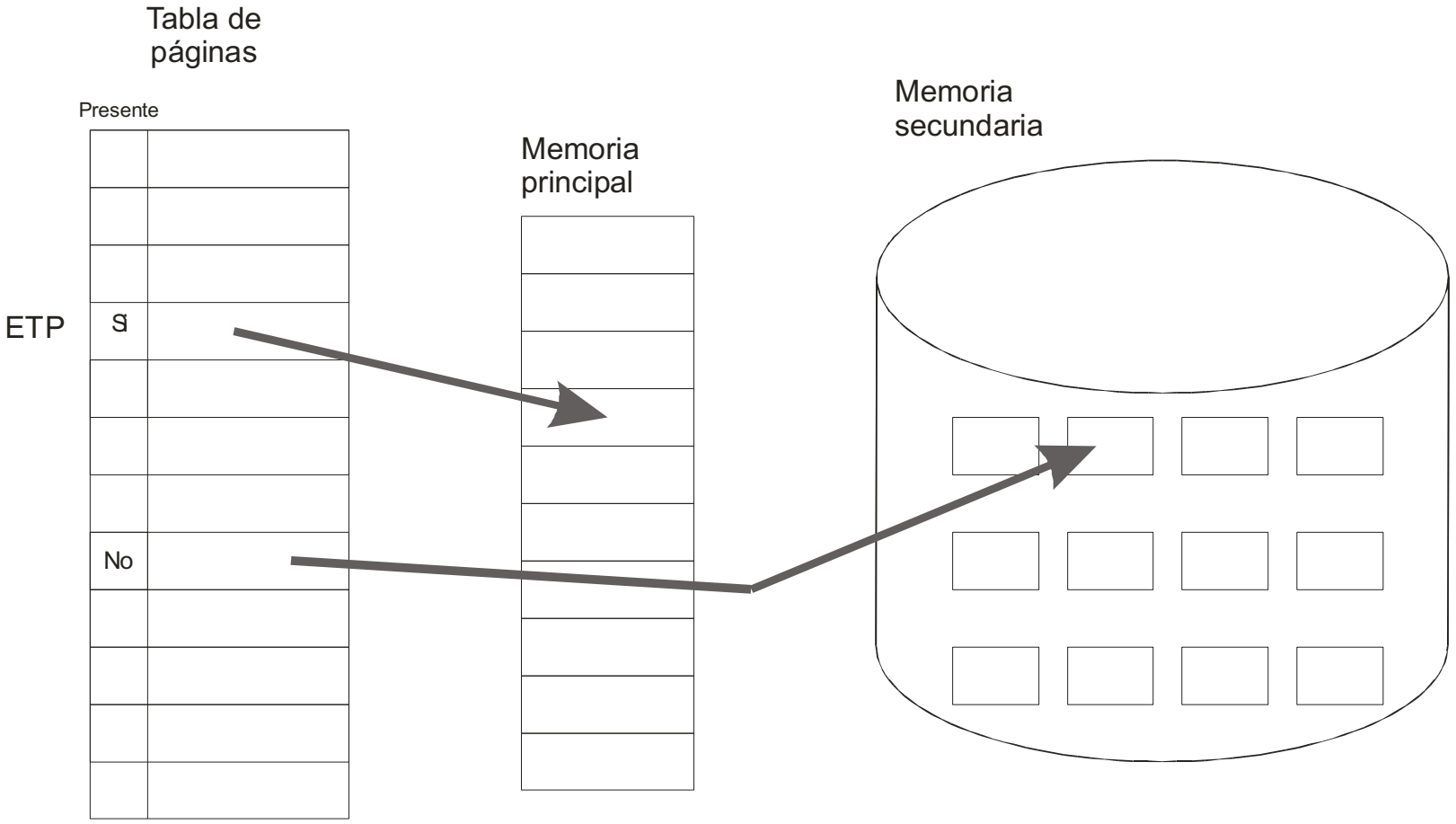
Page Table

- A page table (PT) is needed per process
  - To link virtual page with physical frame in RAM or Swap.
  - PT needed for translation.
- PT: one entry per logical page, including:
  - Logical address of the page
  - Protection bits.
  - Absent/present bit to know is the page is in:
    - RAM memory → Page frame
    - Swap -> block of the swap device

Other info in PT entry

- *Copy-on-write*
- Not paged (locked in physical memory).
- Cache not active.
- Fill to 0.
- Fill from a file.

Page table (II)



- If the page is not in RAM => **page fault** (*trap* to OS).

On-demand paging

- Processes don´t load pages in memory during setup.
- OS creates the TP with all entries as non-resident
- As page faults occur, pages are loaded in RAM.

- Advantages:
  - Only references pages loaded in MP.
  - Faster process creation, as useless pages are not loaded.
  - Enhances RAM usage.
- Method implemented in all current OS:
  - UNIX.
  - LINUX.
  - Windows.

Page fault management

- Hw → *trap* "page fault"
- Control Unit:
  - Saves process state and jumps to Page Fault Management routine.
- OS:
  - Receives page number as parameter from the HW.
  - Access allowed (PT info)? If not allowed, a signal is sent and process is killed.
  - Looks free memory frame. If not available executes replacement algorithm.
  - If page in frame "dirty" → write to swap, and activation of another process.
  - Free frame is locked in RAM, read data and process activation.
  - When I/O interruption signals page read, PT is updated and frame is unlocked in RAM.
  - Processes changed to "ready".
  - Execution "restarts" in the instruction producing the page fault.

Execution of a LINUX program

- Code example.

```
#include <math.h>
#include <stdio.h>

main()
{
   double x = 30;
   double res;
   void *p;

   p= sbrk(40000);

   res = sin(x);
   printf("res = %f \n", res);
}
```

- Libraries:
  - `libc.so , libm.so`

Strace

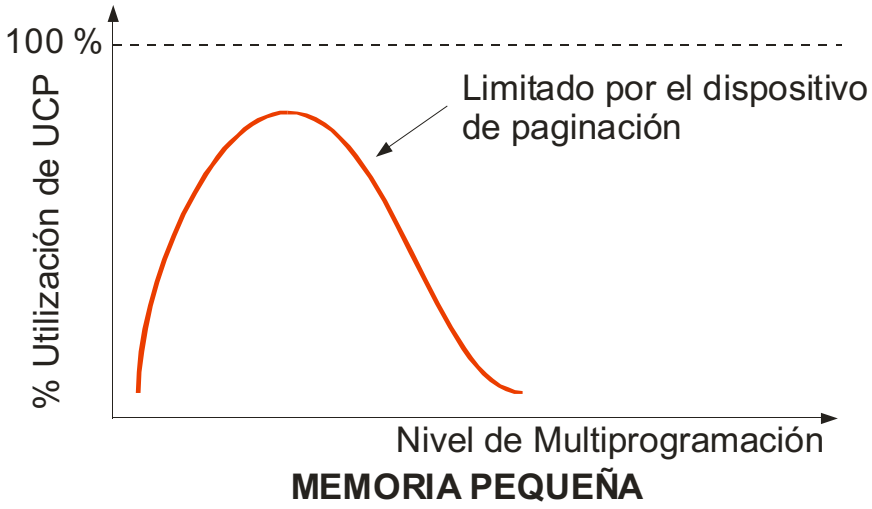
- Command **`strace`**
  - `open("/lib/libm.so", O_RDONLY)`
  - `mmap()` => code projection.
  - `open("/lib/libc.so", O_RDONLY)`
  - `mmap()` => code projection.
  - Start process execution
- File `/proc/PID/`**`maps`**

direccion	perms	offset	dev	nodo-i	
08048000-08049000	r-xp	00000000	08:11	630795	Code
08049000-0804a000	rw-p	00000000	08:11	630795	Data without IV
0804a000-08054000	rwxp	00000000	00:00	0	Dynamic data
40000000-4000a000	r-xp	00000000	08:01	30602	ld.so
4000a000-4000b000	rw-p	00009000	08:01	30602	
40010000-40028000	r-xp	00000000	08:01	30614	libm.so
40028000-40029000	rw-p	00017000	08:01	30614	
40029000-400ba000	r-xp	00000000	08:01	30606	libc.so
400ba000-400c2000	rw-p	00090000	08:01	30606	
400c2000-400ce000	rw-p	00000000	00:00	0	
bfffc000-c0000000	rwxp	ffffd000	00:00	0	Stack

Page replacement

- Goal: reducing number of page faults.
- Algorithms:
  - Optimal, FIFO, NRU, Second chance, Clock.
  - LRU
    - Most frequently used.
- Page buffering.
- Paging demon.
  - Activated everytime pages are needed.
  - Goal: to have free pages in advance.

Trashing



- Solution: reducing the multiprogramming index, suspending one or more processes

OPERATING SYSTEMS:

Lesson 10:
Virtual Memory

Jesús Carretero Pérez
David Expósito Singh
José Daniel García Sánchez
Francisco Javier García Blas
Florin Isaila